The FEniCS Project: Automation and Algorithms for Finite Element Methods

Robert C. Kirby
The University of Chicago
Sandia Livermore Nat'l Lab

Acknowledgments

- Matt Knepley (ANL), Anders Logg (TTI-C), Kevin Long (SLNL), Ridg Scott (UC), Andy Terrel (UC)
- DOE ECPI Program
- SLNL/CSRI

Outline

- Motivation/Overview
- FEM basis functions: FIAT
- Optimizing element matrices: FErari

Motivation

Incompressible NSE

$$\mathbf{u} \cdot \nabla \mathbf{u} + \tfrac{1}{\rho} \nabla p - \nu \Delta \mathbf{u} = 0$$
$$\nabla \cdot \mathbf{u} = 0$$

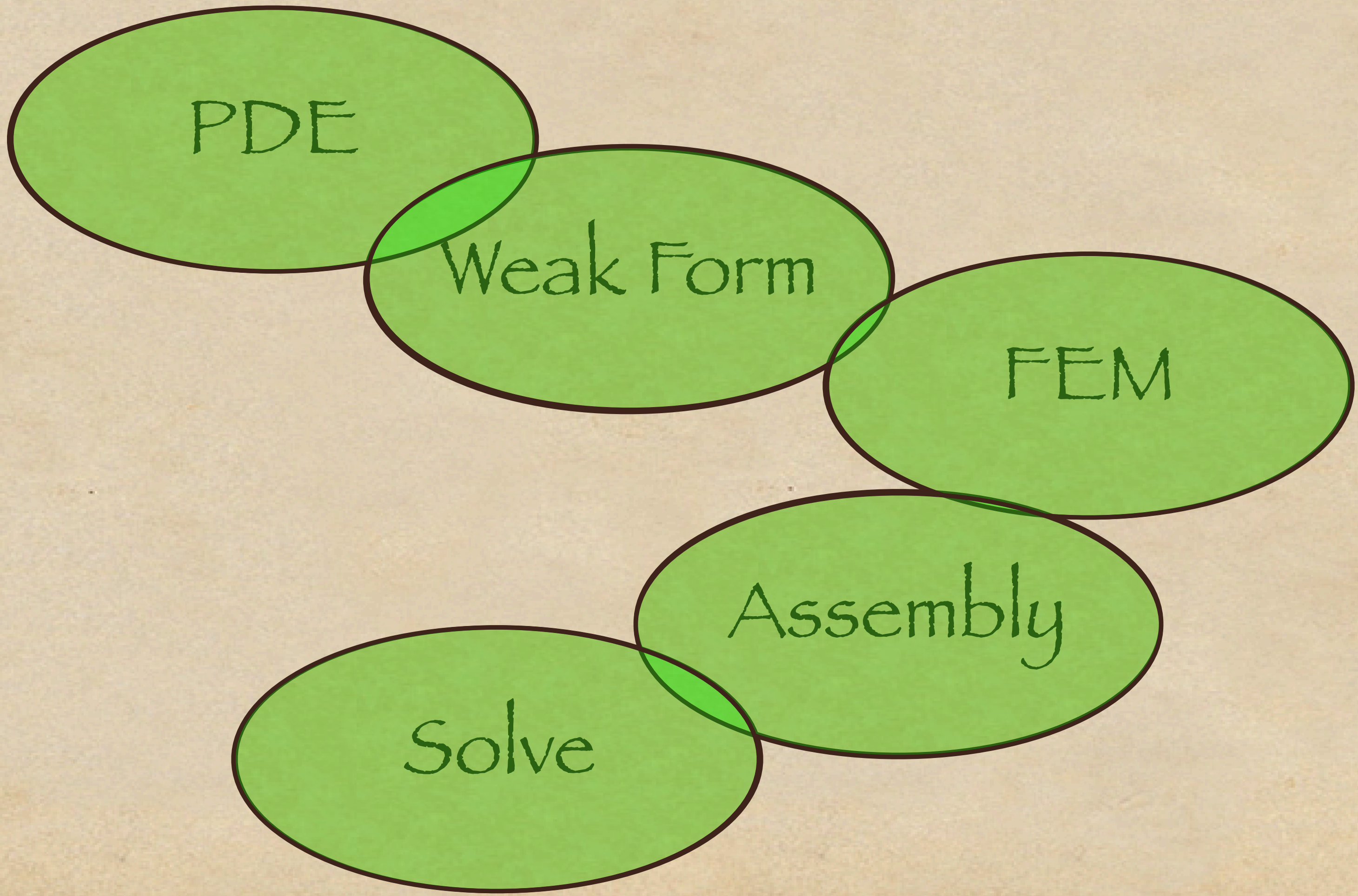
Boussinesq (heat transfer coupled)

$$\mathbf{u} \cdot \nabla \mathbf{u} + \tfrac{1}{\rho} \nabla p - \nu \Delta \mathbf{u} = \beta (T - T_0) g \hat{\mathbf{k}}$$
$$\nabla \cdot \mathbf{u} = 0$$
$$\rho c_p \mathbf{u} \cdot \nabla T - \nabla \cdot (k \nabla T) = 0$$

Problems

- Each piece is tough
- Coupling black boxes?
- Changing order of approximation?
- Functional versus optimal
- More terms: MHD? Viscoelastic?
- Inversion?

The Great Chain of FEing



Enumerative approach

- List all the forms/elements you want
- Implement
- Hope you don't need more
- Difficult to extend due to:
    - Cost to implement single form
    - Cost to make different forms communicate

Grammatical approach

- Specify abstraction for forms/elements
- Generate efficient code
- Benefits:
    - Efficiency, Reliability, Integrability, Extensibility

What do we have?

- Parallel solver libraries (e.g. PETSc, Trilinos)
- Emerging technologies:
    - Sundance, FFC, PETSc
    - FIAT
- Math

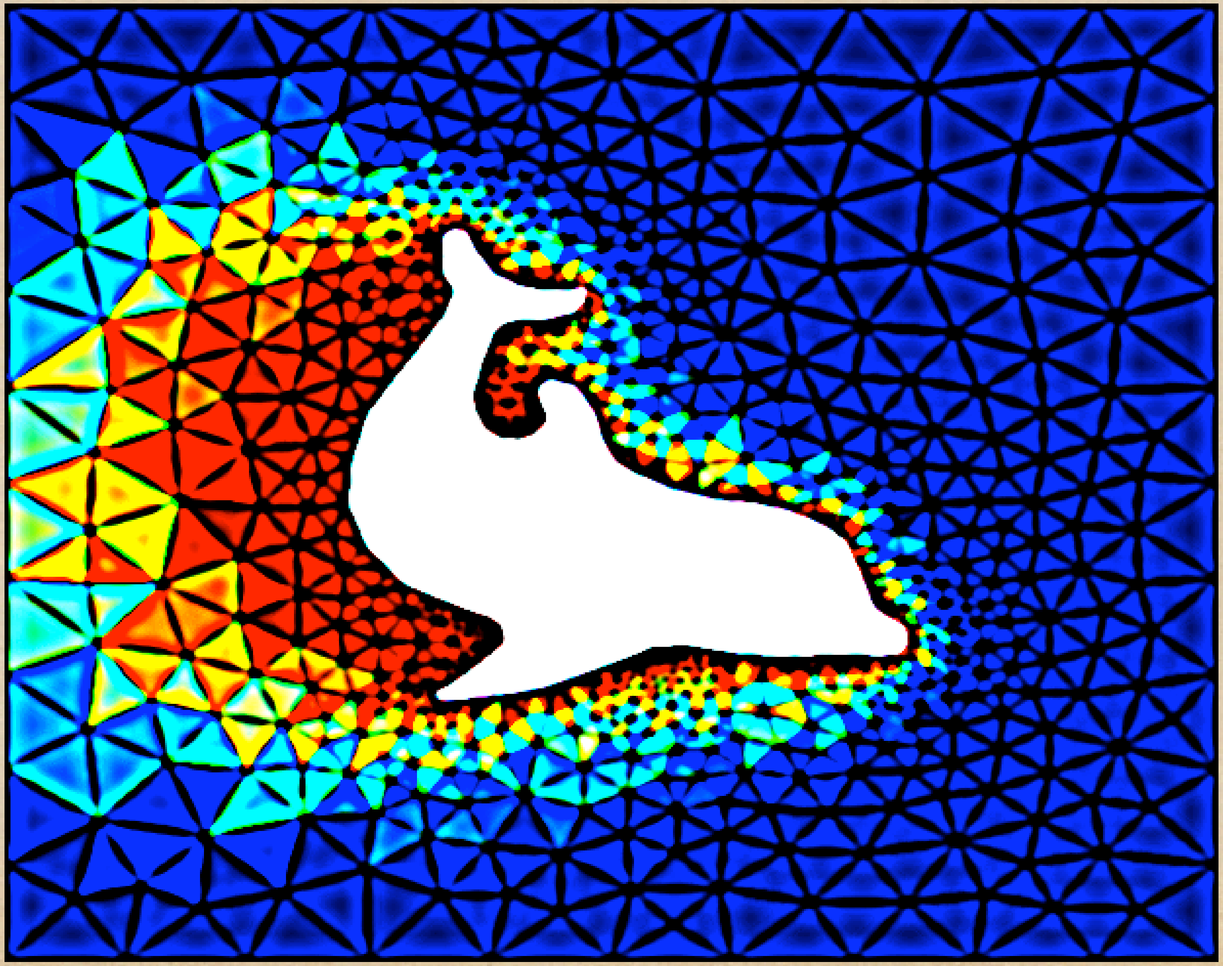
Example: FFC

- FEniCS Form Compiler (Anders Logg)
- Variational form --> DOLFIN code
- Generate a mapping from mesh to matrix.
- PETSc linear algebra
- See also Sundance/Trilinos

FFC Code

```
scalar = FiniteElement("Lagrange", "triangle", 1)
vector = FiniteElement("Lagrange", "triangle", 1, 2)
v  = BasisFunction(scalar) # test function
u1 = BasisFunction(scalar) # value at next time step
u0 = Function(scalar)      # value at previous time step
w  = Function(vector)      # convection
f  = Function(scalar)      # source term
k  = Constant()            # time step
c  = Constant()            # diffusion
a = v*u1*dx + 0.5*k*(v*w[i]*u1.dx(i)*dx + c*v.dx(i)*u1.dx(i)*dx)
L = v*u0*dx - 0.5*k*(v*w[i]*u0.dx(i)*dx + c*v.dx(i)*u0.dx(i)*dx) + v*f*dx
```

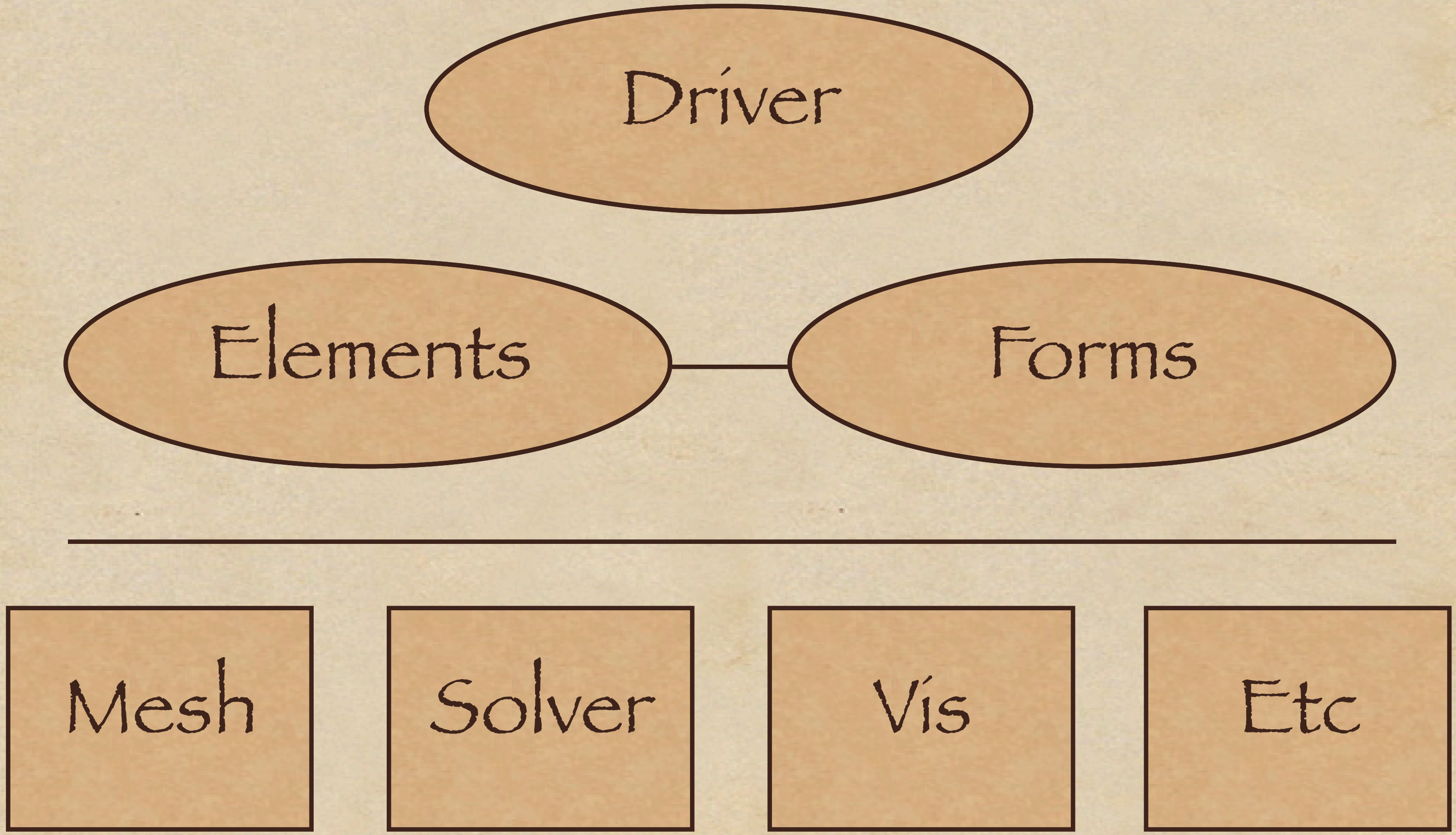
Pretty picture



What else do we need?

- Generating FE basis functions:
  - H^1, H(div), H(curl), high order
  - Assembly
- Parallel (comes from mesh and algebra)
- Optimizing element matrices

High-level view



General Finite Elements

- Underappreciated problem!!
- General order: limits family
- General spaces: limits order
- Need a “representation theory”
- This is called…“linear algebra”

A constructive approach to nodal bases (FIAT)

- What is a finite element?
- What is a nodal basis?
- How do we compute one?

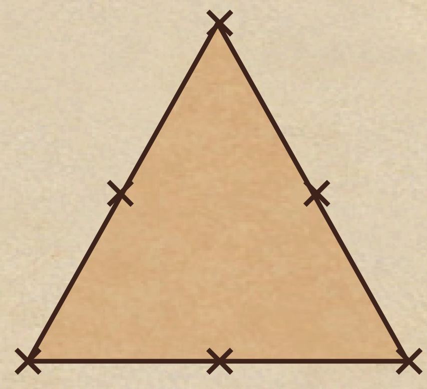
Ciarlet: defining a finite element

A finite element is a triple (K,P,N):

- K a domain with p.w. smooth boundary
- P a f.d. function space (polynomials)
- N a collection of "nodes"
  - linear mappings from P to reals
  - span P'

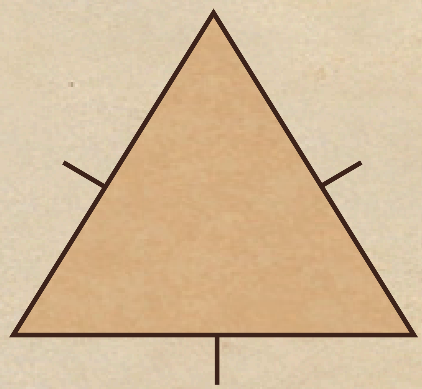
Example: Lagrange

- K: a triangle
- P: Quadratic polynomials
- N: evaluation at 6 points



Example: Raviart-Thomas

- K: a triangle
- P: $(P_k)^2 + xP_k$
- N: normal component at edge midpoints



Nodal bases

The nodal basis is a set $\{\psi_i\}_{i=1}^{\dim P}$

- Basis for P
- Satisfies $n_i\left(\psi_j\right) = \delta_{i,j}$
- Enables interelement continuity
- Formulae? (Hierarchical? Rectangular?)

Computing nodal basis

Start with "prime basis" $\{\phi_i\}_{i=1}^{|P|}$

- Computable formulae
- Stable
- Black box
- For P_k, use orthogonal polynomials

Change of basis

Build Vandermonde matrix $V_{i,j} = n_i(\phi_j)$

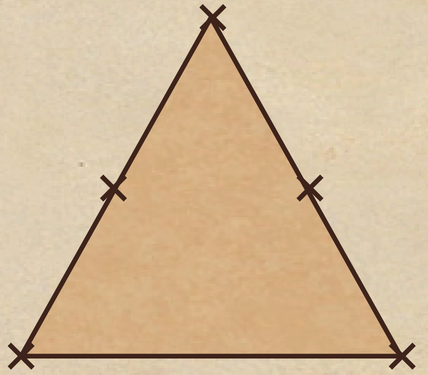
- Columns of inverse are expansion coefficients of nodal basis
- Not as bad as the “real” Vandermonde matrix
- Need code abstractions for functionals

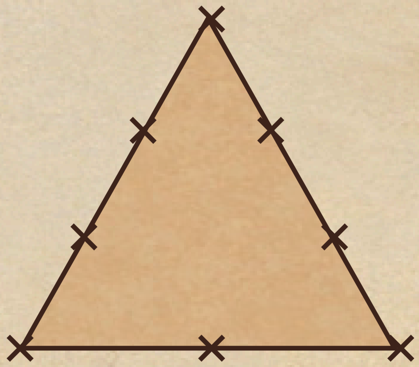
$P \neq P_k$

- p-refinement
- BDFM elements
- Arnold-Winther elements
- Divergence-free spaces
- Can't use (directly) the orthonormal spaces!

Constrained Lagrange

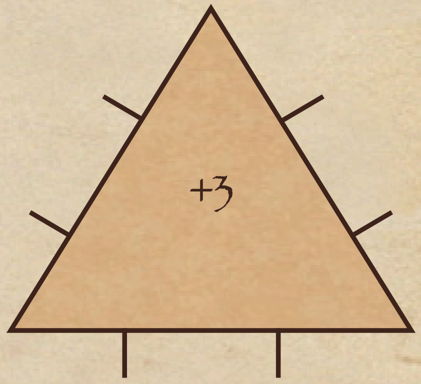
- K: a triangle
- P: Quadratic polynomials, linear on bottom edge
- N: evaluation at 5 points





Example: BDFM

- K: a triangle
- $P = \{p \in (P_k)^d : u \cdot n \in P_{k-1}(\partial K)\}$
- N: normal component on edges, plus some others inside



Building a prime basis

Suppose we have $P \subset \bar{P}, \{\ell_i\}_{i=1}^d$ with

- $\ell_i : \bar{P} \to \mathbb{R}$ linear functional
- $P = \cap_{i=1}^d \mathrm{null}(\ell_i)$
- $\{\bar{\phi}_i\}_{i=1}^{|\bar{P}|}$ a prime basis for $\bar{P}$

Building a prime basis

- Build matrix: $L_{i,j} = \ell_i\left(\bar{\phi}_j\right)$
- Compute SVD: $L = U_L \Sigma_L V_L^t$
- Prime basis: $\phi_j = V_{k,j+|\bar{P}|-|P|}\bar{\phi}_k$
- Bramble-Hilbert (Dupont-Scott)

Implemenation (FIAT)

- Python (C++ coming online)
- All polynomials and functionals are represented as vectors (Riesz Rep Thm)
- Building Vandermonde, constraint matrices is level 3 BLAS
- SVD, inversion done by LAPACK

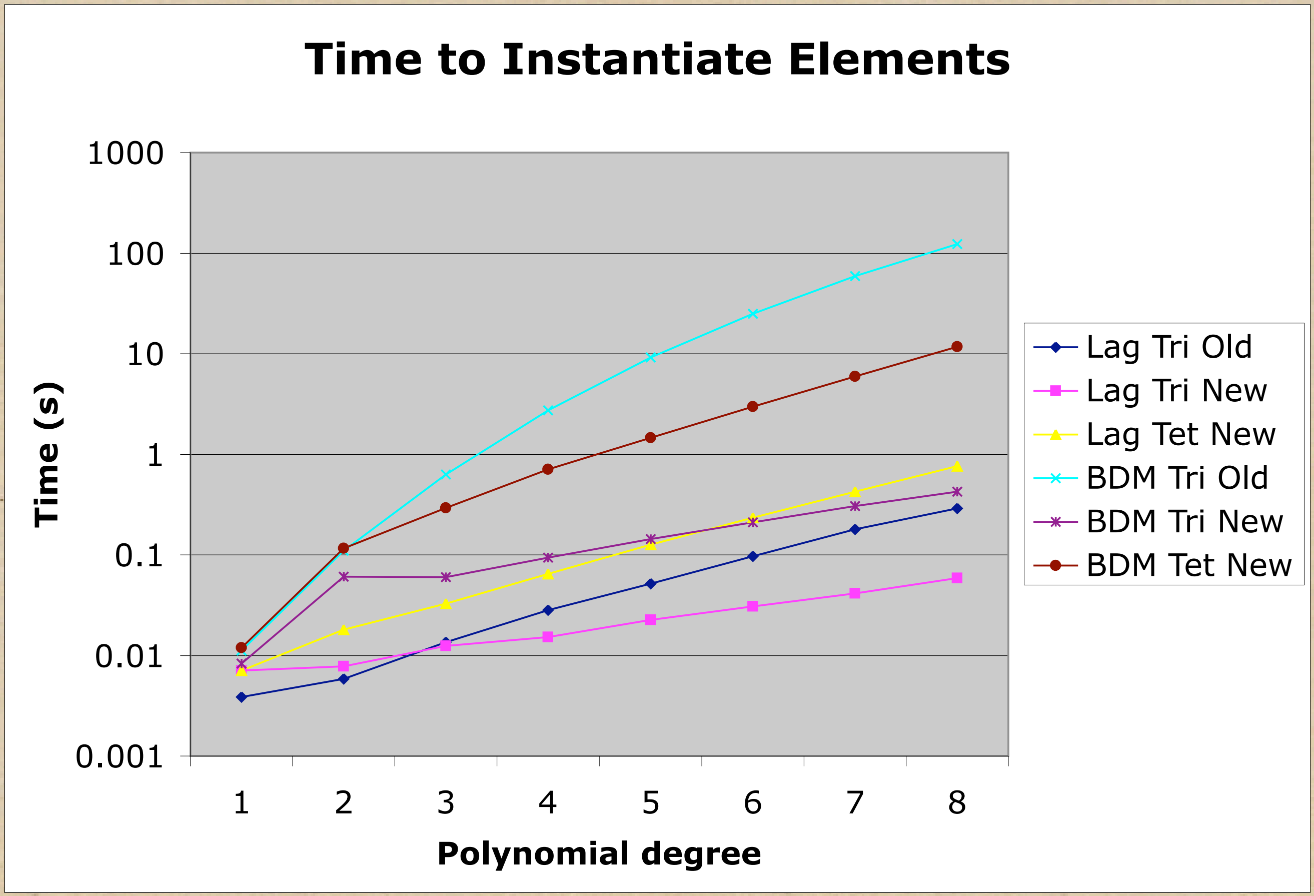
Implementation, cont'd

- Supports simplicial elements
- Lagrange, BDM, Hermite currently in place (one class for each does all the shapes -- see Knepley's incidence relations)
- Available LGPL (www.fenics.org)

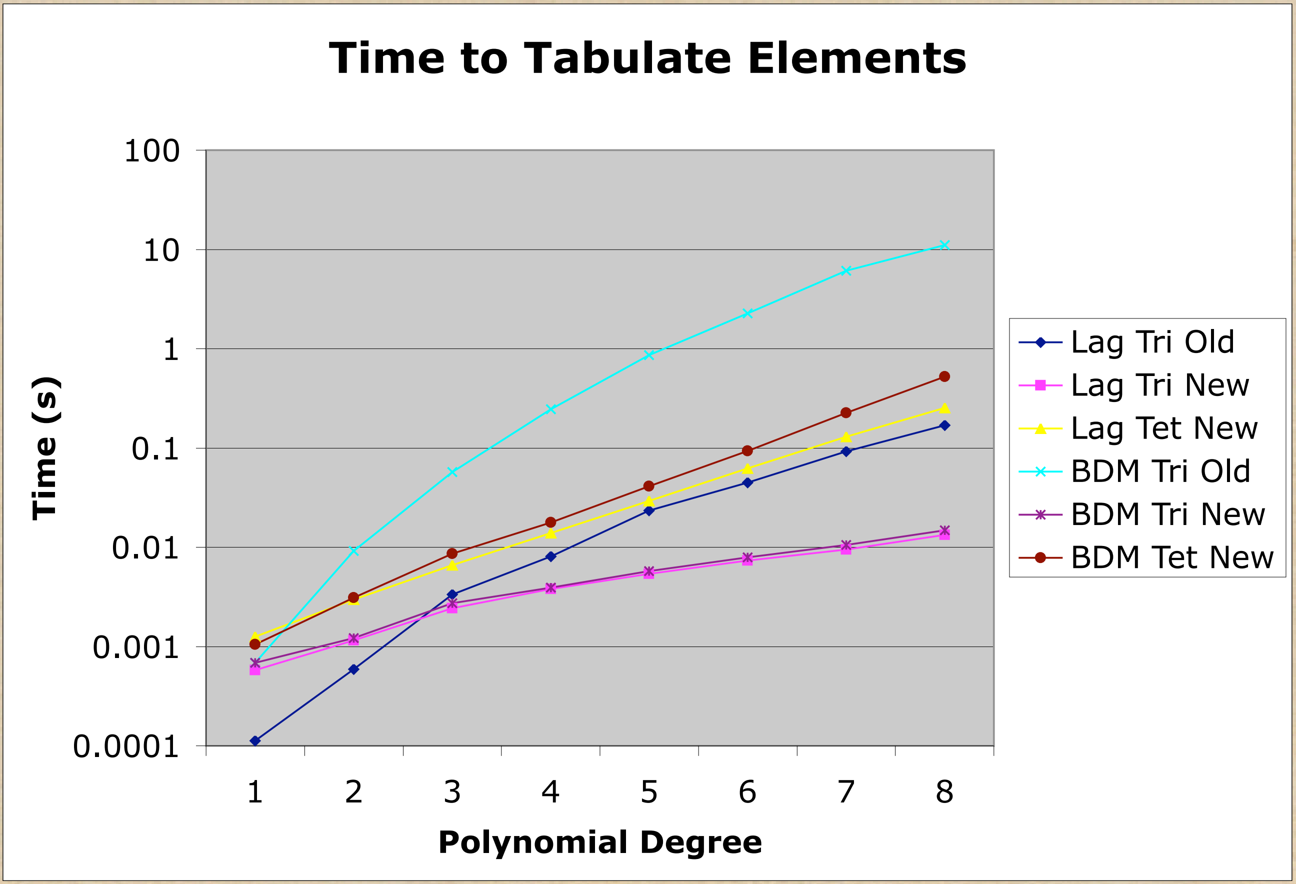
Level 3 BLAS

$$\begin{gathered}
p = p_i \phi_i \\
\ell(p) = p_i \ell(\phi_i) \\
R(p)_i = p_i \\
R'(\ell)_i = \ell(\phi_i) \\
\ell(p) = \ell_i p_i \\
V_{i,j} = \ell_{i,k} p_{j,k}
\end{gathered}$$

Performance



Performance, cont'd



Optimizing form evaluation

- When does it matter?
  - Steady versus unsteady
  - Linear versus nonlinear
  - How good is the solver?
  - Matrix or matrix-free?
- Matters most when there is frequent reconstruction

Local form for Poisson

$$
\begin{aligned}
K^e_{i,j} &= \int_e \nabla \psi_i \cdot \nabla \psi_j \, dx \\
&= \int_{\hat{K}} J^{-t} \left(\hat{\nabla} \psi_i \right) \cdot J^{-t} \left(\hat{\nabla} \psi_j \right) \, d\hat{x}
\end{aligned}
$$

- How fast can we compute, given basis functions?
- How fast can we do action?
- Approach should generalize to other forms!!

Algorithms for LSM

Method	Cost per entry in K
Quadrature	O(k^d)
Precomputation	d^2
Optimal	???

Precomputing Poisson

$$K^e_{i,j} = \mathbf{K}_{i,k,m,m'} G^e_{m,m'}$$

$$\mathbf{K}_{i,j,m,m'} = \int_{\hat{K}} \frac{\partial \psi_i}{\partial \xi_m} \frac{\partial \psi_j}{\partial \xi_{m'}} \mathrm{d}\xi \quad G^e = \frac{J^{-t} J^{-1}}{|J|}$$

$$(Ku)^e_i = \mathbf{K}_{i,j,m,m'} \left(G^e_{m,m'} u^e_j \right)$$

- Similar for other forms
- "Reference element" & "geometry"
- Compute K offline at "compile time"

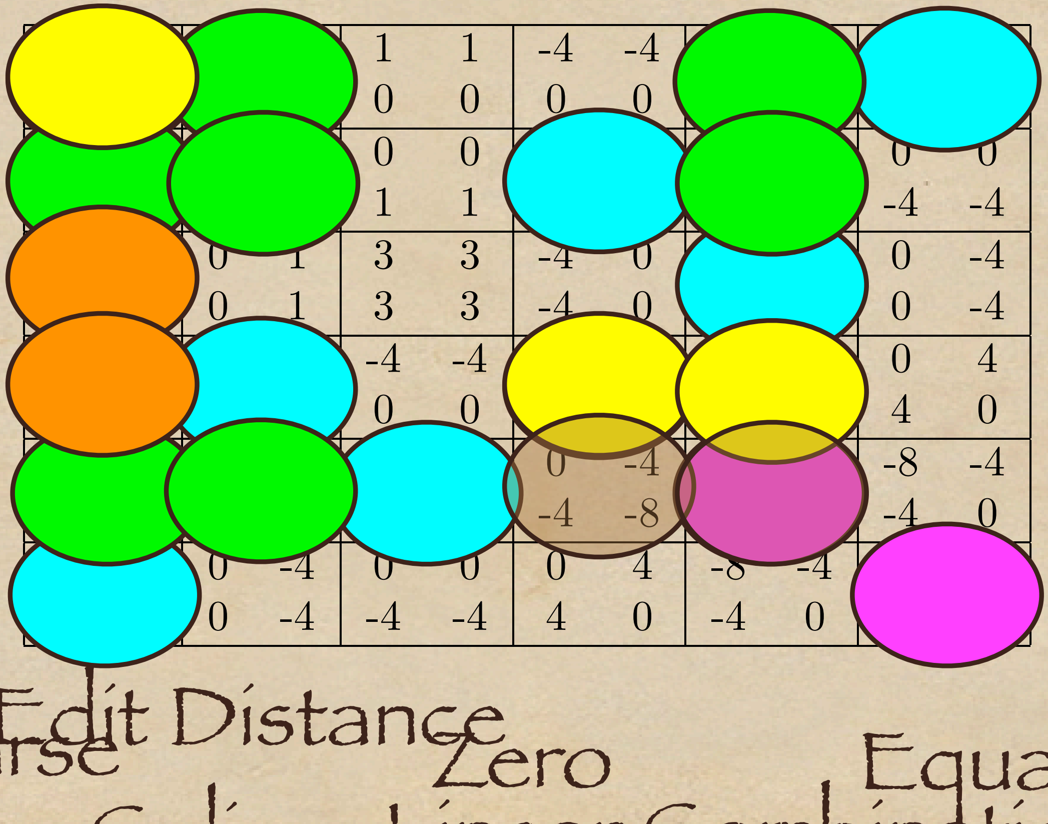
Algorithm

- For each e
    - Get G^e
        - For each $1 \leq i, j \leq |P|$
            - Compute $\mathbf{K}_{i,j} : G^e$
        - Insert block into global matrix

Goal

- Minimize time spent doing all the tensor contractions (whether for matrix-full or matrix-free)
    - Phrase as level 3 BLAS (dense)
    - Find a lower-flop computation (sparse)

K for Poisson (k=2,d=2)



Symmetry

- Only compute triangular part
- Dot products go from d^2 to choose(d+1,2) (G symmetric)
- Preserves other dependencies
- Infer from AST?

Optimization problem

- Given a collection of V of n vectors of length d
- Find a fast algorithm for computing dot products of all elements of V with any arbitrary vector of length d
- Similar for all multilinear forms & actions
- This is compile-time optimization

Comments

- V random ==> nd multiply-add pairs
- But V comes from algebraic structure
- Finding the true optimum is intractible

A topological approach

- Impose distance relations on V
- d(u,v) small ==> u.g is easy to compute from v.g
- Need relations of general arity (linear combinations)

Some binary relations

- equality ($e(u,v) = 0$ or d)
- colinearity ($c(u,v) = 0,1$ or d)
- Hamming distance
- These are all "complexity reducing"
- The min over CR-relations is CR

Using binary relations

- Assume a CR relation r (WLOG)
- Build a graph (V,E)
  - weight of (u,v) is r(u,v)
  - Sparse or dense graph
- Want a traversal of the graph that is minimal cost

Minimum spanning tree

- Starts from root node
- Every node has a parent
- Sum of edge weights is minimal over all spanning trees
- Optimal computation under relation r
- How good is r?

Code generation

- Annotate edges in graph with type of dependencies
- Breadth-first search of MST ==> code generation
- Computes straight-line code
  - array read/write, multiply & add

Results of Poisson MST

- Down to 1-2 flops per entry
- Dominant cost is writing the answer!

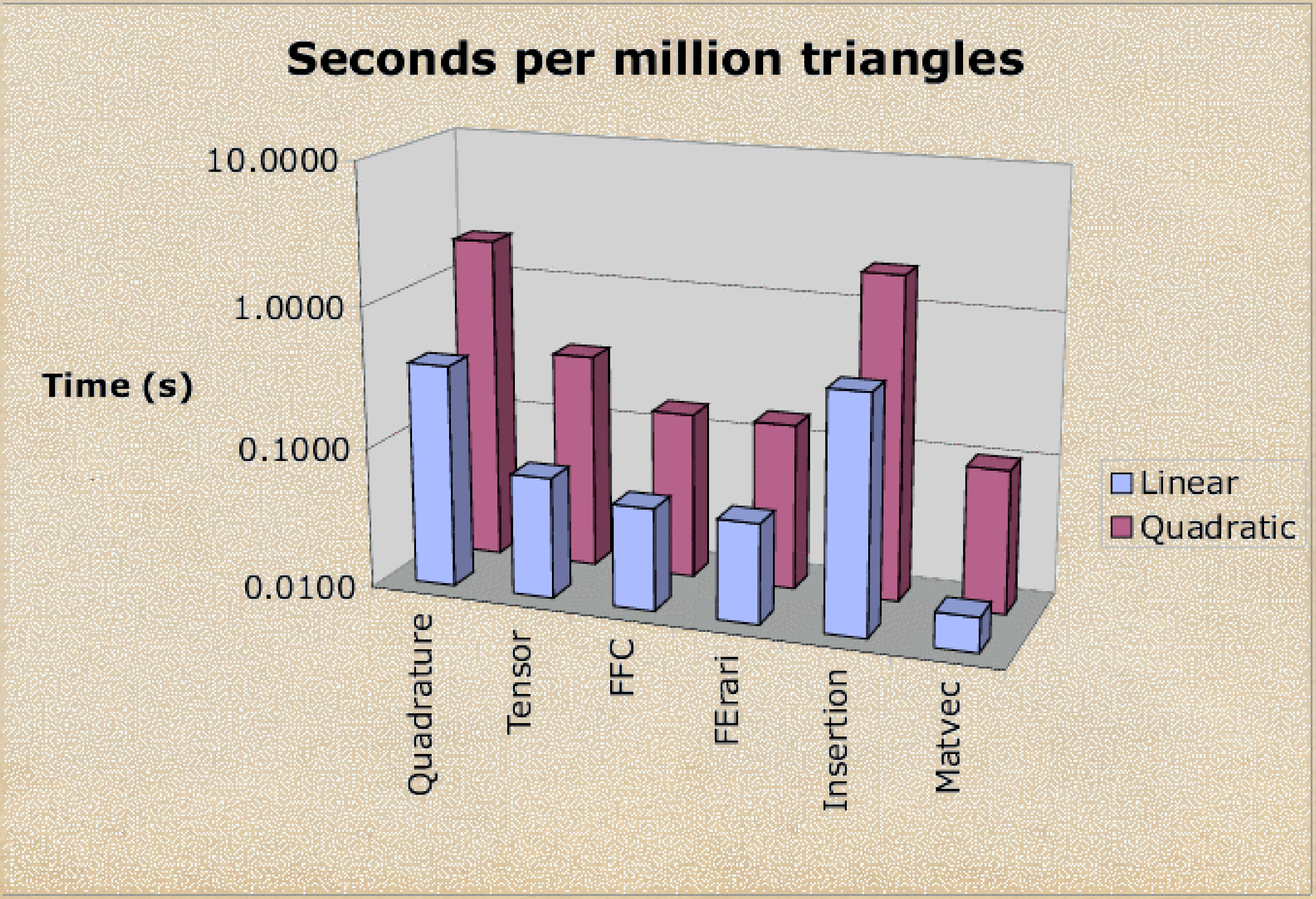
triangles

degree	n	m	nm	MAPs
1	6	3	18	9
2	21	3	63	17
3	55	3	165	46

tetrahedra

degree	n	m	nm	MAPs
1	10	6	60	27
2	55	6	330	101
3	210	6	1260	370

Timing results



Build versus solve

- GMRES/AMG requires three iterations
- AMG build/apply dominates run-time
- Optimized code still gives overall 5-10% speedup
- Geometric MG? Matrix-free?

Results for Advection

- Constant coefficient
- Similar reduction in operation count

triangles

degree	n	m	nm	MAPs
1	9	2	18	4
2	36	2	72	22
3	100	2	200	59

tetrahedra

degree	n	m	nm	MAPs
1	16	3	48	9
2	100	3	300	35
3	400	3	1200	189

Variable coefficient

- Consider weighted Laplacian

$$a_w(v, u) = \int_\Omega w(x)\nabla v(x) \cdot \nabla u(x)\mathrm{d}x$$

- Coefficient projected into FE space
- Much more complicated operator!
  - Rank 5 tensor
  - “Geometry” is rank 3 (includes w)

Tensors

$$A^0_{i\alpha} = \int_E \Phi_{\alpha_1}(X) \frac{\partial \Phi_{i_1}(X)}{\partial X_{\alpha_2}} \frac{\partial \Phi_{i_2}(X)}{\partial X_{\alpha_3}} \mathrm{d}X$$

$$G^\alpha_e = w_{\alpha_1} \det F'_e \frac{\partial X_{\alpha_2}}{\partial x_\beta} \frac{\partial X_{\alpha_3}}{\partial x_\beta} = w_{\alpha_1} \left(G^L\right)^{(\alpha_2,\alpha_3)}_e$$

Three approaches

- Form "full" G, optimize contractions with rank three tensors
- Partially reduce geometry (optimize this), densely contract with coefficient
- Partially reduce coefficient (optimize this), densely contract with geometry

Results on tetrahedra

- Contracting coefficient first wins
- Base costs are 240, 3300, 25200
- Much more flops per memory operation

	G_e			$(G^L)_e$ first			w_k first		
degree	MST	additional	total	MST	additional	total	MST	additional	total
1	108	6*4	132	27	10*4	67	9	10*6	69
2	1650	6*10	1710	693	55*10	1234	465	55*6	795
3	14334	6*20	14454	7021	210*20	11221	7728	210*6	8988

Relations of general arity

- e.g. Linear combinations $t(u,v,w) = 2$ or d
- Can modify MST algorithm
    - Isn't a tree (hypertree)
    - Finding true optimum NP-hard?

Ongoing work

- Algorithms:
  - How quickly can we identify hyperplanar relations?
  - What's the extension of the MST
- Experiments
  - Matrix action (preconditioning?)

Conclusion

- Automation: Generality, Efficiency, Reliability, etc etc etc
- Requires new mathematical applications, interpretations of existing mathematics.